Vi, Java, Ant, Junit 自学报告

Vi

其实初级实训的时候就应该开始要学习使用 vi 来编写代码的了，但是当时看到文档就感觉这个编辑器十分的难用，后来就放弃使用了。想不到在今年中级实训还是要求使用 vi，看来想逃都逃不掉了。于是便试着用一下。

果然，一进去就不知道该怎么办了。想输入文本也不行。原来这是 Normal 模式，输入的是操作文本的指令。要想输入文本，就要按 i 键进入 Insert 模式，当然也有其他按键进入 Insert 模式。然后开始编写一些简单的 java 代码。他跟其他的编辑器一样，对于大多数的编程语言都有语法高亮的特点。但是还是十分不习惯，因为它不像其他编辑器一样，随着你的输入，会给出单词候选项用于自动补全代码。上网查询，输入 Ctrl + N 或者 Ctrl + P 就能达到目的了。

若没有保存代码，是不可以直接退出的。防止工作的丢失，这也算是比较人性化的设定。所以要养成习惯，先保存再退出。用了一天下来，感觉它比 gedit 强大，但逊色于 sublime-text，跟 eclipse 相比就更不用说了。可能是我还没掌握 vi 的操作的精髓吧。使用了 vi，就可以脱离鼠标的束缚，只需要键盘的操作，因此编辑文本的速度也会加快。尽管如此，但是我觉得，编程更多的是用心编程，在代码中表现自己的思维过程。在掌握 vi 的命令之前，需要花时间去想命令，去查命令。实在会打断思考，降低编程的效率。在网上也能看到，vi 的学习曲线十分陡峭。的确是让人望而生畏啊。总的来说，十分难学。不过对于编辑器的选择还是见仁见智吧，适合自己的编辑器才是最好的编辑器。

Java

暑假的时候也有自学 java，但是也只学到了一些皮毛。学习 java 的时候，要跟学过的 C++和 C#联系起来，其实它们之间有许多共同点。所以在语法的理解上面并没有太大的困难。例如 Java 也有继承，但是跟 C++不一样，Java 不支持多继承，也有 C#的 Interface。当然，一些方法也比 C++要强大，例如字符串跟数字之间的转换等等。跟 C++相比最大的不同，就是 Java 没有了指针这个概念了。没有了指针，就不能直接对底层进行内存操作了，但是指针的使用不当也会导致程序的崩溃。另外 Java 具有垃圾回收的功能。New 出来的对象不需要手动 delete，会被自动回收，这就防止了内存泄漏。对于程序员来说也是福音。所以，使用 java 来编程的时候，会有一种安全感。此外跟 eclipse 搭配使用效果更佳。

需要写一个简单的计算器程序。因此我们还要学习 java 的 GUI 编程。Java 的 GUI 功能主要集中在 awt 和 swing 两个包中，awt 是 GUI 底层包，而 swing 包是高层的封装，更容易移植。其实要写一个界面，代码量也不算大。参照网上编写界面的例子就能自己试着写一个界面出来了。然后写了一个跟 wiki 上的简单计算器差不多的界面。写完界面之后就写底层的实现。这里主要是关于事件触发。事件触发主要是通过实现一些接口，如 ActionListener，然后实现 actionPerformed 方法即可。那么当每次事件触发的时候（如按钮按下），就会去调用一次 actionPerformed 方法。写完简单的计算器之后，看见有很多同学又把简单的计算器扩展了其他的功能，不过我觉得还是先把当前的程序继续完善，使我的程序更加健壮。例如我在输入框输入英文字母，又或者是把 0 作为除数时候，不会导致程序的崩溃，而是返回 NaN 的结果。因此，我对于一些代码段，也做了异常处理。后来发现，即使用了 double 来保存结果，也会有精度丢失，如 1 – 0.1，它的答案却不是 0.9，因此我学习使用 BigDecimal

类型来保存计算数据，它能用来保存非常大的数，十分实用。

总得来说，java 用起来十分顺手，相信以后也还有大把机会会用到 java 的，所以我会继续深入学习 java。

ANT

它是一个 Java 项目的自动化部署工具。开始时看到教程，其作用跟之前生成 C 和 C++的 makefile 很相似，但 ant 还是跟它有所不同。Makefile 是相当于帮你在终端输入一系列的指令，而 ant 不仅仅是输入指令那么简单。它有许多 task，而每个 task 又有其特定的属性。不过常用的那几个 task 也是需要记下来的，如 mkdir, delete, javac, java 等等。试着给昨天的简单计算器程序写一个 build.xml，发现 ant run 却无法运行。上网查询，发现这是因为在 ant 中是调用 antClassLoader 来运行的，而我们在终端里使用 java 指令的时候，程序是调用 jvmClassLoader 来运行的。所以可以通过设置 java 这个 task 里面的一个 fork 属性，将其设为”true”就能正常运行了。但是对于 Junit 的测试生成，却无法达到我想要的目的，即通过 ant 来进行自动化单元测试。这里可能就设计到比较复杂的 task 了吧。

我觉得，学习如何写 ANT 还有一个好方法。我们可以用到 eclipse 来帮助我们生成 build.xml。首先在项目名称那里右键，选择 Export，然后选择 Ant Buildfiles，然后点击 next, finish，最后就生成了一个 build.xml。通过查看 eclipse 自动生成的 xml 文件，可以对如何写 build.xml 有更深入的了解。尤其是涉及到有 Junit 的时候。上网很难查到关于 ant 部署 junit 的资料。因此这也算是一个比较好的参考。

JUnit

感觉 Junit 也是似曾相识的。就是跟在初级实训的时候所使用过的 gtest 差不多，通过断言来检查每一个函数的正确性。但跟 gtest 相比也有其强大之处。例如 Junit 可以 ignore 掉一些未完全实现的函数，或者给函数一个特定的执行时间以防止发生死循环，还可以进行参数化的测试。若在 eclipse 下部署 JUnit，就显得十分简单。但是不用 eclipse 的话就要输入一些比较麻烦的命令，例如要将 classpath 指定到 Junit 所在的目录，因为不知道如何把 Junit 添加到环境变量中，所以每次都要加上 –cp .:……/Junit-4.9.jar，而将 junit 添加到 java 类库也不仅仅是把 jar 文件直接复制到 lib 文件夹里这么简单。

每次完成一个函数，就应该对该函数进行测试以保证其正确性。一般的做法就是自己写一个主函数，然后想一些测试样例去调用它，然后观察其输出是否达到预期。这种做法虽然可行，但是比较麻烦。利用 JUnit 就可以比较方便地对自己的代码进行单元测试，而且编写测试代码的效率也比较高。